Z80 assembler operation

Output Files Produced

There are a number of file types produced from the assembler, although some of them will only be produced if no errors are generated from the assembly.

The file types produced all have the same name as the source file, but have differently generated extensions as follows:

.BIN	This file contains the binary machine code produced from the program if a single file has been assembled, or if several files are linked together. It is a straight binary dump of the code with no other information included.
.INX	This is the Intel hex version of the machine code produced by the program.
.DBG	This is the debug information file which is produced for the monitor. It is always produced and indicates the start and finish of the code and includes data areas and label values. This is an ASCII file which can be read and modified by a word processor.
.LST	This is the list file. It contains a list of errors if present, compilation time, and for reference the code start and end addresses. It will only be produced if the file parameters have been set up to produce a list file, see Set File Parameters option
.ZBJ	This file is produced if the linker is used to create a binary file from one or more input modules which can be linked. This is a binary file whose contents can only be read with the 'readobj' program.
.EQU	This file is produced by the assembler or linker and consists of every label in the program with an EQU statement suitable for being Included into another assembly.

Source Code

Source Code is stored in the assembler file which is an ASCII text file and usually has the extension .ASS.

The assembler uses standard Z80 mnemonics throughout with the exception of the instruction:

```
EX AF,AF'
```

which is written in the assembler without the the inverted comma, so it becomes:

```
EX AF,AF
```

The source code must be in capital letters (except for labels), and may contain any spaces, tabs, or newlines required to make the text more legible.

Comments are introduced by a ';' character, and any information on the line following the ';' is ignored. Comments may be on a line of their own, or may follow any source code on a line.

Example: The following is a short example of code designed to multiply the DE and HL registers and return the 16 bit result in the HL register. Overflows are ignored.

```
;
; ROUTINE TO MULTIPLY THE HL AND DE REGISTERS
;
; RESULT IS RETURNED IN THE HL REGISTER

             STARTC &1800      ; Code Start Address
             ENDC LASTB        ; Code End Address
             ORG &1800         ; Set The Origin
MULTHLDE:    LD A,16           ; A REGISTER IS A LOOP COUNTER
             LD C,L
             LD B,H            ; STORE OPERAND IN BC
             LD HL,0           ; HL HOLDS TOTAL
MULLOOP:     BIT 0,E           ; TEST IF ADD REQD
             JR Z,NOADD
             ADD HL,BC         ; ADD IN THE RESULT
NOADD:       SLA C             ; BC=BC*2
             RL B
             RR D              ; DE=DE/2
             RR E
             DEC A
             JR NZ,MULLOOP     ; LOOP 16 TIMES
             RET

LASTB:       EQU $
```

Labels

Labels may be up to 10 characters in length, and may contain any alphabetic or numeric character, but must start with a letter. Labels are case insensitive.

Labels may be defined on a line in any of 3 ways, they may be defined with a '.' character preceding the label, with a ':' character after the label, or in lower case letters - any characters starting a line in lower case are assumed to be part of a label. For example the following lines all define the label LOOP :

```
LOOP:       LD BC,89

.LOOP       LD BC,89

loop        LD BC,89
```

Labels may not contain the strings 'IX' or 'IY' anywhere within them, and must not be the same as any of the following keywords:

```
'A','B','C','D','E','F','H','I','L','R'

'AF','BC','DE','HL','SP'

'M','NC','NZ','P','PE','PO','Z'
```

The following are all examples of illegal labels

```
7UP               - Starts with a number
PO                - Reserved Keyword
LOOP_1      - Contains non-alphanumeric
FIXUP             - Contains the string 'IX'
TOTALRETURN       - More than 10 characters
```

The following are all examples of legal labels:

```
MULTLOOP
S7UP
DIVHLDE
LOOP1
TOTALRETRN
SUBCOUNT
```

Notes for use of labels with the linker

When the linker is to be used there are 2 types of label.

The first is described as relocatable and is a program reference which may change as the position of the module changes during the link.

The second type is described as an absolute label because it does not change during the link. Absolute labels are set up with the EQU directive, and are normally used to set up constants or to fix the position of machine code variables in RAM.

An expression may contain references to any number of absolute labels, but may contain only one relocatable label. This is because relocatable labels change in value, and if more than one were to be given in an expression then the object file would have to hold the complete expression for evaluation during link.

A relocatable label may be part of an expression but only if it is used with a constant offset. For example `LOOP+9`, `LOOP-3+2`, `3*2+LOOP-9`, are all valid as they all evaluated to a constant offset from the label `LOOP`. However `LOOP*2` or `LOOP/3` are invalid because the expression is not an offset from `LOOP`.

The following are valid source code lines:

```
JP START+3          ; START may be relocatable or absolute
LD A,(9+LX-LY)      ; Provided that only one of LX or LY is a
                    relocatable label.
```

The same label may be defined in different modules, in this case the first defined label is taken and the subsequent labels are ignored. This allows routines in modules to be bypassed by rewriting the routine and calling it the same name in a module which is earlier in the linker list of files. Whenever a duplicate label is found a warning will be issued.

Absolute labels may also be defined more than once, however:

a) a label must not be relocatable in one module but absolute in another.

b) an absolute label if defined in more than one module must have the same absolute value in all modules in which it is defined.

Expressions

An expression may be used anywhere in the source code that a number would be used. The expression evaluator has no facilities for operator precedence or for bracketing expressions.

Numbers may be specified in binary, decimal, or hex, or as an ASCII character. The default for the assembler is decimal.

Binary numbers are specified with a '%' character preceding the number.

```
Example:    %100001     ; 33 in decimal
```

Decimal numbers are specified in normal decimal form. This is the default for the assembler.

```
Example:    33
```

Hex numbers are specified with an '&' character preceding the hex number which may have characters A-F in addition to the normal numbers.

```
Example:    &21         ; 33 in decimal
```

Characters may be specified so that the ASCII value of the character is returned. This is done by surrounding the character with single quotes.

```
Example:    '!'         ; 33 in decimal
```

The current value of the PC may be specified by using the '$' character. It is not permissible to use this if the file is being assembled for use with the linker, as the value of the PC will change during the link process.

Anywhere in an expression that a number is used, a label may be used instead.

The operators are:

+	Addition
-	Subtraction
*	multiplication
/	division
>	right shift
<	left shift
&	bitwise and
|	bitwise or
^	bitwise xor
~	complement

```
Examples of valid expressions:

1+2*3             ; value 6.
```

```
LOOP-3      ; value of label LOOP-3.
-1                ; value -1 decimal, FFFF in hex.
'A'+&10           ; value 81.
&10>2             ; value &04.
~8                ; value &FFF7
```

Assembler Pseudo ops / Directives

There are a number of source code directives which affect the assembly or the output produced, and which are not part of the standard Z80 instruction set. These are detailed below.

ENDC	Defines code save end address when assembling a single file
EQU	Inserts a single defined byte at a location.
INCLUDE	Include another file in the assembly at this point
DEFB	Inserts a single defined byte at a location.
DEFW	Inserts a defined word at a location.
DEFM	Inserts an ASCII message at a location.
DEFS	Leaves space for a defined number of bytes.
ORG	Defines the code origin.
STARTC	Defines code save end address when assembling a single file

DEFB

This is used to insert bytes at the current point in the assembly. The DEFB directive is followed by a list of one or more bytes which will be inserted.

```
Example:    DEFB 0,1,'A',2,3,LOOP
```

This will insert the bytes 00,01,65,2,3, and the lower byte of the value of the label LOOP.

DEFW

This is used to insert words at the current point in the assembly. The DEFW directive is followed by a list of one or more bytes which will be inserted in low byte-high byte format.

```
Example:    DEFW &ED67,1,LOOP
```

This will insert the bytes &67,&ED,01,00, and the value of the label LOOP.

DEFS

This inserts a number of bytes at the current point in the assembly. The DEFS statement is followed by the number of bytes to insert, and then optionally separated by a comma, the byte to be inserted. If no byte is specified then the area is filled with byte 00.

```
Example:    DEFS &200
```

This will insert 512 byte 0's at the current point in the assembly.

```
Example:    DEFS 64,' '
```

This will insert 64 spaces.

DEFM

This directive is used to insert a text message at the current position in the assembly. The argument to the directive is an ASCII text string.

```
Example: DEFM "Syntax in Command !"
```

This will insert the bytes 'S','y','n','t','a','x',' ','i','n',' ','C','o','m','m','a','n','d',' ','!'.

ENDC

This defines the last byte of the binary file produced which is to be saved. If this command is not specified then the last byte assembled is the last byte saved.

```
Example: ENDC 2047
```

This will ensure that all the object code from the first byte (specified by STARTC) to the byte at address 2047 is saved in the binary file.

This directive is ignored if the -o option has been specified to produce an object file.

EQU

This has no effect on the assembly, but sets up a label to equal to a value. The label value is absolute, that is it will not be modified by the linker. EQU is normally used to set up constants, and to fix locations which the linker must not change such as RAM based machine code variables.

```
Example: BUFHEAD EQU 2050
```

This sets the label BUFHEAD to the value 2050.

INCLUDE

This directive is used to bring in another source file at the current point in assembly. The source file is assembled, and then the existing file continues from the first line after the INCLUDE directive. INCLUDE files may be nested to a depth of 9, and the assembler will search for the files not only in the current directory, but then in every path specified in the environment variable INCLUDE. The main use for the directive is to provide header files which contain EQU directives so that every module which is linked to make up a program may have a common data area.

```
Example: INCLUDE "MONITOR.H"
```

This line includes the source file MONITOR.H which is supplied with the package, and which defines all the labels required for the use of the monitor routines.

ORG

This sets the value of the program counter so that assembly continues from a different address. The value supplied should not be a forward reference, that is it should be numeric, or should be set to the value of a label which has already been declared.

```
Example: ORG 2048
```

This sets assembly on following lines to start at address 2048.

This directive is ignored if the -o option has been specified to produce an object file.

STARTC

This sets the start address for the code to be saved in the binary file. The binary file consists of code saved from the address specified in STARTC to the address specified in ENDC. If STARTC is not specified then it is set to 0.

```
Example: STARTC &1000
```

This saves code in the binary file starting from address 4096.

This directive is ignored if the -o option has been specified to produce an object file.

"Illegal" Opcodes

The Z80 has a number of codes which are not described in the Zilog documentation. The assembler supports some of the more useful of these codes. They are almost guaranteed to operate correctly, it is rumoured that some very early versions of the Z80 had bugs which prevented them operating and Zilog left them out of the documentation, the bugs were later corrected. These codes were widely used in many games on home computers in the 1980's.

Index Half Registers

The unshifted opcodes which act on the H or the L registers- all those codes which do not start with &CB or &ED -will operate on the upper or lower halves of the IX and IY registers if preceded with a &DD or &FD code. The assembler supports these and the registers are referred to as IXH, and IXL for the upper and lower halves of the IX register, and IYH and IYL for the IY register. For example:

```
LD IXH,7        ; loads the top 8 bits of IX with the value 7
LD A,IYL        ; loads A with the bottom 8 bits of IY
```

The codes act on the flags in the same way as instructions with H or L, in most cases adding 4 T-states and one byte to the instruction length.

Preceding the rotate, shift, bit, set and reset instructions (codes which start with &CB) with &DD or &FD create all sorts of interesting effects - some instructions cause other registers to be shifted, some cause strange values to be loaded into other registers, in fact anything but the IX and IY registers are affected ! For this reason the assembler will throw out instructions which start with &CB such as RL IXH.

SLL

The SLL instruction shifts it's operand left one place, replacing bit 0 with a 1, and the carry becomes what was bit 7 of the operand. The instruction fills in the "missing" instruction from &CB,&30 to &CB&37 and act in the same way as the other shifts e.g. SRL

Command Reference, Use of Windows

Menu Options

File Menu
Edit Menu
Compile Menu
Project Menu
Window Menu
Help Menu

Windows

Project Windows
Edit Windows

Project Windows

A project is a collection of files which are to be assembled and linked into a complete machine code image. A project file (with the extension .ZPJ) holds the list of all the files in the project and the options associated with each file and the complete project. It also stores the positions of file windows when they were last open, and also any other files that were open when the project was last open.

To open a project the Project|Open Project menu option is used. An existing project may be opened, or if it is desired to create a new project then the name of a project which doesn't exist may be entered.

Items may be added or deleted from the project by using the Project|Add Item and ~~Project|Delete~~ Item Whenever a file is part of a project then a window will be open for that file to be edited, closing the window will delete the file from the project. Within the project window double clicking on a file name will bring that file editing window to the fore to be edited immediately.

Projects are saved automatically whenever a new project is opened, or whenever the assembler is closed, if a project has not been named the user will be prompted for the name on saving the project.

The project name is used for the output file name in linking, the individual file names are used in assembling.

Edit Windows

Editing windows allow files to be edited.

Editing windows have a status and button bar along the top. The buttons are Save and Print which act like the File|Save and File|Print menu options, the Assemble button which acts like the Compile|Assemble File menu, and the Params button which acts like the Compile|Set File Parameters menu. The status information shows the current line number and character position of the cursor, and whether the file has been saved since it was last modified.

The edit window acts exactly like the standard Windows Notepad application and the key presses are identical, however edit windows automatically indent code when the enter key is pressed. There is one exception to this, pressing Ctrl and F1 together whilst the cursor is on a Z80 opcode (e.g. INC, LD etc.) will bring up the help application with the topic which describes that opcode's format and operands etc.

File Menu

Open

This opens a file for editing using an editing window. The file is not part of the project and will not be linked in with the project (unless, of course, it is included in another file with an INCLUDE statement). However it may be assembled individually. The project file stores a list of files which have been opened and they are reopened whenever the project is opened.

Save

This saves the file which is currently being edited (the window at the top).

Save As

This saves the file which is currently being edited (the window at the top), but allows the file name to be changed.

Save All

This saves all files which are currently open, regardless of whether they are in the project or not.

Insert File

This prompts the user for a file name and inserts that file at the current cursor location in the file which is currently being edited (the window at the top).

Print

This prints the file which is currently being edited (the window at the top). The print out includes the time and date and the file name at the top of the listing. During the print it may be cancelled using the Print Cancel dialog box.

Print All

This prints all files which are currently open, regardless of whether they are in the project or not. The print out includes the time and date and the file name at the top of the listing. During the print it may be cancelled using the Print Cancel dialog box.

Printer Set Up

This brings up the printer set up box for the current default printer, which is the one used by the assembler.

Exit

This closes the assembler, prompting the user to save any files which have been modified since they were last saved.

Edit Menu

Search

This allows the user to search for text in the file, and is equivalent to using Shift and the F3 keys.

Search Again

This repeats the last search to find the next occurrence of the text found using the Search menu option. Note that each open file maintains it's own search text, so that search again will not find text searched for in another file editing window.

Replace

This brings up a dialog box which allows the user to replace occurrences of text with new text. The user may enter text to search for and text to replace it with, this latter box may be left blank to delete text. If the query replace box is clicked then the user will be prompted on each occurrence of the search text to confirm replacement. If the replace button is clicked only one occurrence of the search text will be replaced, the Replace All button will replace all text until the bottom of the file.

Replace Again

This repeats the last replace to find and substitute the next occurrence of the text found using the Replace menu option. Note that each open file maintains it's own replace text, so that replace again will not find text searched for in another file editing window.

Cut

Removes selected text from the window.

Copy

Copies selected text to the clipboard.

Paste

Inserts text from the clipboard at the current cursor position.

Undo

Undoes the last action on the window, if the last action was a replace all then this will only undo the last replace action, not all of them.

Clear

This clears all text from the window.

Goto

Prompts the user for a row and column position and sends the cursor to that position, provided that it is within the file.

Compile Menu

Set File Parameters

This brings up the set file parameters dialog box which allows the user to enter:

Address	The address of the file in the link. If this is blank then the file will be linked in order with the other files which are being linked. This address has no effect on a standalone assembly, only on a link.
Produce List File	This produces a list file (the same as the assembled file name with a .LST extension) whenever the file is assembled either on it's own or as part of a link.
Case Sensitive	This specifies that the assembly is to be case sensitive - in this case all opcodes and operands must be in upper case, and it is best to leave this option unchecked.

Assemble File

This assembles the current file. See the Assembling and Linking section above for details.

Make Project

This makes the project, assembling only those files which have changed since the project was last made, and linking all the object files. See the Assembling and Linking section for details.

Assemble/Link All

This makes the project, all files in the project are assembled and linked regardless of whether they have changed since the project was last made. This option should be used if include files have changed.

Project Menu

Open Project

This option opens a project. If an existing project is named then this will be opened, if the project does not exist then it will be created. Opening a project automatically saves and closes the existing project.

Add Item

This will add a file to the project. It will be opened for editing, and the editing window for that file will stay open with the project whilst the file is in the project. It is not possible to add a file which is already open for editing, it must be closed first.

Delete Item

This option removes a file from a project so that it will no longer be assembled and linked with the other files in the project.

Window Menu

Tile

This causes all the windows on the assembler desktop to be resized so that they can all be seen at the same time.

Cascade

This arranges all windows on the desktop so that they all lie on top of one another so that the title bars of all windows are visible.

File List

The Window menu contains a list of all windows and any of them can be selected by clicking on the window title under the list of files.

Help

Contents

This brings up the help application for the Z80 assembler with its contents list.

Look up Keyword

Whilst the cursor is on a Z80 opcode (e.g. INC, LD etc.) this will bring up the help application with the topic which describes that opcode's format and operands etc.

The same effect is achieved by pressing Control-F1 with the cursor on the keyword.

About

Displays copyright and version information.

Z80 Assembler/Linker Help

Contents

Registration Information

This program is shareware, should you decide that you wish to use it for serious purposes - in practice any project which results in code used on a Z80 or a simulator, then you must register it.

Registration costs £20.00 or $30.00, and should be made payable to "Robin Abbott" at the following address:

37 Plantation Drive
Christchurch
Dorset

ENGLAND

BH23 5SG

Registration entitles you to a printed manual, technical support and an evaluation copy of the next version of the program free of charge.

To obtain the evaluation copy of the remote symbolic monitor/debugger which links in with debugging information produced by the debugger and which runs on a PC in conjunction with a Z80 monitor send £5 or $10.00 to the same address.

Z80 Instruction Set

The Z80 Instruction set is divided into the following subject areas:

8 bit load group	Instructions which load 8 bit registers and memory locations.
16 bit load group	Instructions which load 16 bit registers and memory locations, includes PUSH and POP
8 bit arithmetic and logical group	Instructions which act arithmetically on 8 bit operands.
16 bit arithmetic and logical group	Instructions which act arithmetically on 16 bit operands.
General purpose arithmetic/CPU Control	Instructions which affect specific registers, or CPU state.
Rotate and Shift Group	Instructions which rotate or shift operands.
Exchange,Block Transfer/Search group	Instructions which move memory around, search memory, and exchange registers.
Bit Set, Reset, and Test Group	Instructions which manipulate individual bits in operands.
Jump Group	Instructions which Jump by modifying PC.
Call and Return Group	Subroutine Operations.
Input and Output Group	Operations which use the Z80 I/O space.

Technical Topics

Z80 Registers	A review of the Z80 registers.
Interrupt Modes	A brief discussion of Z80 interrupts.
Shift Operations	What the various shift operations do.
Symbolic Notation for Tables	The notation used in the tables of assembler mnemonics.

Symbolic Notation used for assembler tables

Symbol	Operation
S	Flag
Z	Flag
P/V	Flag
H	Flag
N	Flag
H&N	Flag
C	Flag
|	The flag is affected according to the result of the operation.
.	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation
X	The flag is a "don't care"
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation
r	Any one of the CPU registers, A,B,C,D,E,H,L.
s	Any 8 bit location for the addressing modes allowed for the instruction.
ss	Any 16 bit location for the addressing modes allowed for the instruction.
ii	Any of the index registers, IX or IY
R	Refresh counter.
n	8 bit value in the range 0-255
nn	16 bit value in the range 0-65535. Lower byte is always written first.
(X)	Contents of address expressed by X.
X<b>	Bit number b of the operand X.

Z80 Registers

The Z80 has the following Registers:

8 Bit Registers

MAIN		**ALTERNATE**		
A		A'		Accumulator (8 bits)
F		F'		Flags Register
B	C	B'	C'	General Purpose and Counter
D	E	D'	E'	General Purpose
H	L	H'	L'	General Purpose, Addressing
I				Interrupt Vector - See Interrupts
R				Refresh Counter

16 bit registers

IX	Index Register (offset, indirect addressing)
IY	Index Register (offset, indirect addressing)
SP	Stack Pointer
PC	Program Counter - Accessed through Jump, Call and Ret instructions

Interrupt Flip Flops

IFF1	Interrupt Enable Flag, 1=Interrupts Enabled, 0=Interrupts Disabled
IFF2	Stores IFF1 during NMI service.
IMFa	\
IMFb	/ Interrupt Mode Flip-Flops, 00=IM0, 01=IM1, 11=IM2.

8 Bit Load Group

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
LD r,R	r=R	..X.X...	01rrrRRR	1	1	4
LD r,N	r=N	..X.X...	00rrr110 nnnnnnnn	1	2	7
LD r,(HL)	r=(HL)	..X.X...	01rrr110	1	2	7
LD r,(IX+D)	r=(IX+d)	..X.X...	11011101 DD 01rrr110 dddddddd	3	5	19
LD r,(IY+D)	r=(IY+d)	..X.X...	11111101 FD 01rrr110 dddddddd	3	5	19
LD (HL),r	(HL)=r	..X.X...	01110rrr	1	2	7
LD (IX+D),r	(IX+d)=r	..X.X...	11011101 DD 01110rrr dddddddd	3	5	19
LD (IY+D),r	(IY+d)=r	..X.X...	11111101 FD 01110rrr dddddddd	3	5	19
LD (HL),N	(HL)=N	..X.X...	00110110 36 nnnnnnnn	2	3	10
LD (IX+D),N	(IX+D)=N	..X.X...	11011101 DD 00110110 36 dddddddd nnnnnnnn	4	5	19
LD (IY+D),N	(IY+D)=N	..X.X...	11111101 FD 00110110 36 dddddddd nnnnnnnn	4	5	19

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
LD A,(BC)	A=(BC)	..X.X...	00001010 0A	1	2	7
LD A,(DE)	A=(DE)	..X.X...	00011010 1A	1	2	7
LD A,(NN)	A=(NN)	..X.X...	00111010 3A nnnnnnnn nnnnnnnn	3	4	13
LD (BC),A	(BC)=A	..X.X...	00000010 02	1	2	7
LD (DE),A	(DE)=A	..X.X...	00010010 12	1	2	7
LD (NN),A	(NN)=A	..X.X...	00110010 32 nnnnnnnn nnnnnnnn	3	4	13
LD A,I	A=I	||X0XI0.	11101101 ED 01010111 57	2	2	9
LD A,R	A=R	||X0XI0.	11101101 ED 01011111 5F	2	2	9
LD I,A	I=A	..X.X...	11101101 ED	2	2	9

			01000111 47			
LD R,A	R=A	..X.X...	11101101 ED	2	2	9
			01001111 4F			

See Symbolic Notation for a description of symbols used in the table.

16 Bit Load Group

Mnemonic	Action	SZ-H-PNC	Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
LD dd,NN	dd=NN	..X.X...	00dd0001 nnnnnnnn nnnnnnnn	3	3	10
LD IX,NN	IX=NN	..X.X...	11011101 DD 00100001 21 nnnnnnnn nnnnnnnn	4	4	14
LD IY,NN	IY=NN	..X.X...	11111101 FD 00100001 21 nnnnnnnn nnnnnnnn	4	4	14

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action	SZ-H-PNC	Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
LD HL,(NN)	L=(NN) H=(NN+1)	..X.X...	00101010 2A nnnnnnnn nnnnnnnn	3	5	16
LD IX,(NN)	IXL=(NN) IXH=(NN+1)	..X.X...	11011101 DD 00101010 2A nnnnnnnn nnnnnnnn	4	6	20
LD IY,(NN)	IYL=(NN) IXH=(NN+1)	..X.X...	11111101 FD 00101010 2A nnnnnnnn nnnnnnnn	4	6	20
LD dd,(NN)	ddL=(NN) ddH=(NN+1)	..X.X...	11101101 ED 01dd1011 nnnnnnnn nnnnnnnn	4	6	20

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action	SZ-H-PNC	Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
LD (NN),HL	(NN)=L (NN+1)=H	..X.X...	00100010 22 nnnnnnnn nnnnnnnn	3	5	16
LD (NN),IX	(NN)=IXL (NN+1)=IXH	..X.X...	11011101 DD 00100010 22 nnnnnnnn nnnnnnnn	4	6	20
LD (NN),IY	(NN)=IYL (NN+1)=IYH	..X.X...	11111101 FD 00100010 22	4	6	20

			nnnnnnnn			
			nnnnnnnn			
LD (NN),dd	(NN)=ddL	..X.X...	11101101 ED	4	6	20
	(NN+1)=ddH		01dd0011			
			nnnnnnnn			
			nnnnnnnn			

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
LD SP,HL	SP=HL	..X.X...	11111001 F9	1	1	6
LD SP,IX	SP=IX	..X.X...	11011101 DD	2	2	10
			11111001 F9			
LD SP,IY	SP=IY	..X.X...	11111101 FD	2	2	10
			11111001 F9			

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
PUSH qq	STACK=qq	..X.X...	11qq0101	1	3	11
PUSH IX	STACK=IX	..X.X...	11011101 DD	2	4	15
			11100101 E5			
PUSH IY	STACK=IY	..X.X...	11111101 FD	2	4	15
			11100101 E5			
POP qq	qq=STACK	..X.X...	11qq0001	1	3	10
POP IX	IX=STACK	..X.X...	11011101 DD	2	4	14
			11100001 E1			
POP IY	IY=STACK	..X.X...	11111101 FD	2	4	14
			11100001 E1			

See Symbolic Notation for a description of symbols used in the table.

Exchange, Block Transfer, Block Search Group

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
EX DE,HL	DE<>HL	..X.X...	11101011 EB	1	1	4
EX AF,AF'*	AF<>AF'	..X.X...	00001000 08	1	1	8
EXX	dd<>dd'	..X.X...	11011001 D9	1	1	4
EX (SP),HL	STACK<>HL	..X.X...	11100011 E3	1	5	19
EX (SP),IX	STACK<>IX	..X.X...	11011101 DD 11100011 E3	2	6	23
EX (SP),IY	STACK<>IY	..X.X...	11111101 FD 11100011 E3	2	6	23

* Note, this assembler uses EX AF,AF not EX AF,AF' for this mnemonic.

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action		Opcode	Bytes	M	T	
		SZ-H-PNC	76543210				
LDI	(DE)=(HL)	..X0X|0.	11101101 ED	2	4	16	PV=0 if
	DE=DE+1		10100000 A0				BC=0
	HL=HL+1						
	BC=BC-1						
LDIR	(DE)=(HL)	..X0X00.	11101101 ED	2	5	21	If BC<>0
	DE=DE+1		10110000 B0	2	4	16	If BC=0
	HL=HL+1						
	BC=BC-1						
	Repeat until BC IS 0						
LDD	(DE)=(HL)	..X0X|0.	11101101 2	4	16	PV=0 if	
	DE=DE-1		10101000 A8				BC=0
	HL=HL-1						
	BC=BC-1						
LDDR	(DE)=(HL)	..X0X00.	11101101 ED	2	5	21	If BC<>0
	DE=DE+1		10110000 B8	2	4	16	If BC=0
	HL=HL+1						
	BC=BC-1						
	Repeat until BC IS 0						

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
CPI	A-(HL)	||x|x||.	11101101 ED	2	4	16
	PV=BC<>0					

	HL-HL+1		10100001	A1				Z=A-(HL)
CPIR	A-(HL)	||x|x||.	11101101	ED	2	5	21	IF BC=0
	HL=HL+1		10100001	A1	2	4	16	IF BC=0
	BC=BC-1							or A=(HL)
CPD	A-(HL)	||x|x||.	11101101	ED	2	4	16	
	PV=BC<>0							
	HL-HL-1		10101001	A9				Z=A-(HL)
CPDR	A-(HL)	||x|x||.	11101101	ED	2	5	21	IF BC=0
	HL=HL-1		10111001	B9	2	4	16	IF BC=0
	BC=BC-1							or A=(HL)

See Symbolic Notation for a description of symbols used in the table.

8 Bit Arithmetic and Logical Group

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
ADD A,r	A=A+r	‖x‖xV0‖	10**000**rrr	1	1	4
ADD A,N	A=A+N	‖x‖xV0‖	11**000**110 nnnnnnnn	2	2	7
ADD A,(HL)	A=A+(HL)	‖x‖xV0‖	10**000**110	1	2	7
ADD A,(IX+D)	A=A+(IX+D)	‖x‖xV0‖	11011101 DD 10**000**110 dddddddd	3	5	19
ADD A,(IY+D)	A=A+(IY+D)	‖x‖xV0‖	11111101 FD 10**000**110 dddddddd	3	5	19

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action		Opcode
		SZ-H-PNC	76543210
ADC A,S	A=A+C+S	‖x‖xV0‖	**001**
SUB S	A=A-S	‖X‖Xv1‖	**010**
SBC A,S	A=A-C+S	‖x‖xV1‖	**011**
AND S	A=A&S	‖X‖XP00	**100**
OR S	A=A‖S	‖X‖XP00	**110**
XOR S	A=A^S	‖X‖XP00	**101**
CP S	A-S	‖X‖XV1‖	**111**

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
INC r	r=r+1	‖X‖XV0.	00rrr**100**	1	1	4
INC (HL)	(HL)=(HL)+1	‖X‖XV0.	00110**100**	1	3	11
INC (IX+D)	(IX+D)++	‖X‖XV0.	11011101 DD 00110**100**	3	6	23
INC (IY+D)	(IY+D)++	‖X‖XV0.	11111101 FD 00110**100**	3	6	23
DEC M	M=M-1					

See Symbolic Notation for a description of symbols used in the table.

General Purpose Arithmetic and CPU Control Groups

Mnemonic	Action		Opcode	Bytes	M	T	
		SZ-H-PNC	76543210				
DAA	A=Packed BCD	‖X‖XP.‖	00100111 27	1	1	4	
CPL	A=~A	..X‖X.‖.	00101111 2F	1	1	4	One's comp.
NEG	A=0-A	‖X‖XV1‖	11101101 ED 01000100 44	2	2	8	Negative
CCF	C=~C	..XXX.0‖	00111111 3F	1	1	4	Complement C
SCF	C=1	..X0X.01	00110111 37	1	1	4	Set C to 1
NOP	NO OPERATION	..X.X...	00000000 00	1	1	4	
HALT	HALT CPU	..X.X...	01110110 76	1	1	4	Wait for INT
DI	IFF=0	..X.X...	11110011 F3	1	1	4	Disable Int
EI	IFF=1	..X.X...	11111101 FB	1	1	4	Enable Int
IM 0	Set IM 0	..X.X...	11101101 ED 01000110 46	2	2	8	
IM 1	Set IM 1	..X.X...	11101101 ED 01010110 56	2	2	8	
IM 2	Set IM 2	..X.X...	11101101 ED 01011110 5E	2	2	8	

See Symbolic Notation for a description of symbols used in the table.

16 Bit Arithmetic Group

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
ADD HL,SS	HL=HL+SS	..XXX.0|	00ss1001	1	3	11
ADC HL,SS	HL=HL+SS+C	||XXXV0|	11101101 ED 01ss1010	2	4	15
SBC HL,SS	HL=HL-SS-C	||XXXV1|	11101101 ED 01ss0010	2	4	15
ADD IX,SS	IX=IX+SS+C	..XXX.0|	11011101 DD 01ss1001	2	4	15
ADD IY,SS	IY=IY+SS+C	..XXX.0|	11111101 FD 01ss1001	2	4	15
INC SS	SS=SS+1	..X.X...	00ss0011	1	1	6
INC IX	IX=IX+1	..X.X...	11011101 DD 00100011 23	2	2	10
INC IY	IY=IY+1	..X.X...	11111101 FD 00100011 23	2	2	10
DEC SS	SS=SS-1	..X.X...	00ss1011	1	1	6
DEC IX	IX=IX-1	..X.X...	11011101 DD 00101011 2B	2	2	10
DEC IY	IY=IY-1	..X.X...	11111101 FD 00101011 2B	2	2	10

See Symbolic Notation for a description of symbols used in the table.

Rotate and Shift Group

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
RLCA	Shift	..X0X.0|	00000111 07	1	1	4
RLA	Shift	..X0X.0|	00010111 17	1	1	4
RRCA	Shift	..X0X.0|	00001111 0F	1	1	4
RRA	Shift	..X0X.0|	00011111 1F	1	1	4
RLC r	Shift	||X0XP0|	11001011 CB	2	2	8
			00**000**rrr			
RLC (HL)	Shift	||X0XP0|	11001011 CB	2	4	15
			00**000**110			
RLC (IX+D)	Shift	||X0XP0|	11011101 DD	4	6	23
			11001011 CB			
			dddddddd			
			00**000**110			
RLC (IY+D)	Shift	||X0XP0|	11111101 FD	4	6	23
			11001011 CB			
			dddddddd			
			00**000**110			
RL m	Shift	||X0XP0|	**010**			
RRC m	Shift	||X0XP0|	**001**			
RR m	Shift	||X0XP0|	**011**			
SLA m	Shift	||X0XP0|	**100**			
SRA m	Shift	||X0XP0|	**101**			
SRL m	Shift	||X0XP0|	**111**			
RLD	Shift	||X0XP0.	11101101 ED	2	5	18
			01101111 6F			
RRD	Shift	||X0XP0.	11101101 ED	2	5	18
			01100111 67			

See Symbolic Notation for a description of symbols used in the table.

Bit Set, Reset, and Test Group

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
BIT B,rR	Z=~r<B>	X|X1XX0.	11001011 CB 01bbbrrr	2	2	8
BIT B,(HL)	Z=~(HL)<B>	X|X1XX0.	11001011 CB 01bbb110	2	3	12
BIT B,(IX+D)	Z=~(IX+D)<B>	X|X1XX0.	11011101 DD 11001011 CB dddddddd 01bbb110	4	5	20
SET B,r	r<B>=1	..X.X...	11001011 CB **11**bbbrrr	2	2	8
SET B,(HL)	(HL)<B>=1	..X.X...	11001011 CB **11**bbb110	2	4	15
SET B,(IX+D)	(IX+D)<B>=1	..X.X...	11011101 DD 11001011 CB dddddddd **11**bbb110	4	6	23
SET B,(IY+D)	(IY+D)<B>=1	..X.X...	11111101 FD 11001011 CB dddddddd **11**bbb110	4	6	23
RES b,m	m<B>=0	..X.X...	**00**			

RES operates on the same operands as SET, note however that the **11** is replaced by a **00**.

See Symbolic Notation for a description of symbols used in the table.

Jump Group

Mnemonic	Action		Opcode	Bytes	M	T	
		SZ-H-PNC	76543210				
JP NN	PC=NN	..X.X...	11000011 C3 nnnnnnnn nnnnnnnn	3	3	10	
JP cc,NN	If CC true then PC=NN	..X.X...	11ccc010 nnnnnnnn nnnnnnnn	3	3	10	

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action		Opcode	Bytes	M	T	
		SZ-H-PNC	76543210				
JR E	PC=PC+E	..X.X...	00011000 18 ffffffff	2	3	12	F=E-2
JR C,E	If C=1 then	..X.X...	00111000 38	2	2	7	No Jump
	PC=PC+E		ffffffff	2	3	12	If Jumped
JR NC,E	If C=0 then	..X.X...	00110000 30	2	2	7	No Jump
	PC=PC+E		ffffffff	2	3	12	If Jumped
JR Z,E	If Z=1 then	..X.X...	00101000 28	2	2	7	No Jump
	PC=PC+E		ffffffff	2	3	12	If Jumped
JR NZ,E	If Z=0 then	..X.X...	00100000 20	2	2	7	No Jump
	PC=PC+E		ffffffff	2	3	12	If Jumped
DJNZ E	B=B-1	..X.X...	00010000 10	2	2	8	If B=0
	If B<>0		ffffffff	2	3	13	If B<>0
	PC=PC+E						

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action		Opcode	Bytes	M	T
		SZ-H-PNC	76543210			
JP (HL)	PC=HL	..X.X...	11101001 E9	1	1	4
JP (IX)	PC=IX	..X.X...	11011101 DD 11101001 E9	2	2	8
JP (IY)	PC=IY	..X.X...	11111101 FD	2	2	8

See Symbolic Notation for a description of symbols used in the table.

Call and Return Group

Mnemonic	Action		Opcode	Bytes	M	T	
		SZ-H-PNC	76543210				
CALL NN	STACK=PC	..X.X...	11001101	3	5	17	
	PC=NN		nnnnnnnn				
			nnnnnnnn				
CALL CC,NN	If CC true	..X.X...	11ccc100	3	3	10	CC false
	then STACK=PC		nnnnnnnn	3	5	17	CC true
	PC=NN		nnnnnnnn				
RET	PC=STACK	..X.X...	11001001 C9	1	3	10	
RET CC	If CC true	..X.X...	11ccc000	1	1	5	CC false
	then PC=STACK			1	3	11	CC true
RETI*	Return from interrupt	..X.X...	11101101 ED 01001101 4D	2	4	14	
RETN*	Return from non-maskable interrupt	..X.X...	11101101 ED 01000101 45	2 2	4 4	14 14	
RST P	CALL P	..X.X...	11ppp111	1	3	11	

See Interrupts for further details of actions on return from interrupt routines.

See Symbolic Notation for a description of symbols used in the table.

Input and Output Group

Mnemonic	Action		Opcode	Bytes	M	T	
		SZ-H-PNC	76543210				
IN A,(N)	A=(N)	..X.X...	11011011 DB	2	3	11	n-
>A0~7
			nnnnnnnn				A-
>A8~A15
IN r,(C)	r=(C)	||X|XP0.	11101101 ED	2	3	12	C-
>A0~7
			01rrr000				B-
>A8~15
INI*	(HL)=(C)	X|XXXX|X	11101101 ED	2	4	16	C-
>A0~7
	B=B-1		10100010 A2				B-
>A8~15
	HL=HL+1						
INIR	(HL)=(C)	X|XXXX|X	11101101 ED	2	5	21	C-
>A0~7
	B=B-1		10110010 B2		B<>0		B-
>A8~15
	HL=HL+1			2	4	16	
	Repeat until B=0				B=0		
IND*	(HL)=(C)	X|XXXX|X	11101101 ED	2	4	16	C-
>A0~7
	B=B-1		10101010 AA				B-
>A8~15
	HL=HL-1						
INDR	(HL)=(C)	X|XXXX|X	11101101 ED	2	5	21	C-
>A0~7
	B=B-1		10111010 BA		B<>0		B-
>A8~15
	HL=HL-1			2	4	16	
	Repeat until B=0				B=0		

* If the result of B-1 is zero the Z flag is set, otherwise it is reset

See Symbolic Notation for a description of symbols used in the table.

Mnemonic	Action		Opcode	Bytes	M	T	
		SZ-H-PNC	76543210				
OUT A,(N)	(N)=A	..X.X.|X	11010011 D3	2	3	11	n-
>A0~7
			nnnnnnnn				A-
>A8~A15
OUT (C),R	(C)=r	..X.X.|X	11101101 ED	2	3	12	C-
>A0~7
			01rrr001				B-
>A8~15
OUTI*	(C)=(HL)	X|XXXX|X	11101101 ED	2	4	16	C-
>A0~7

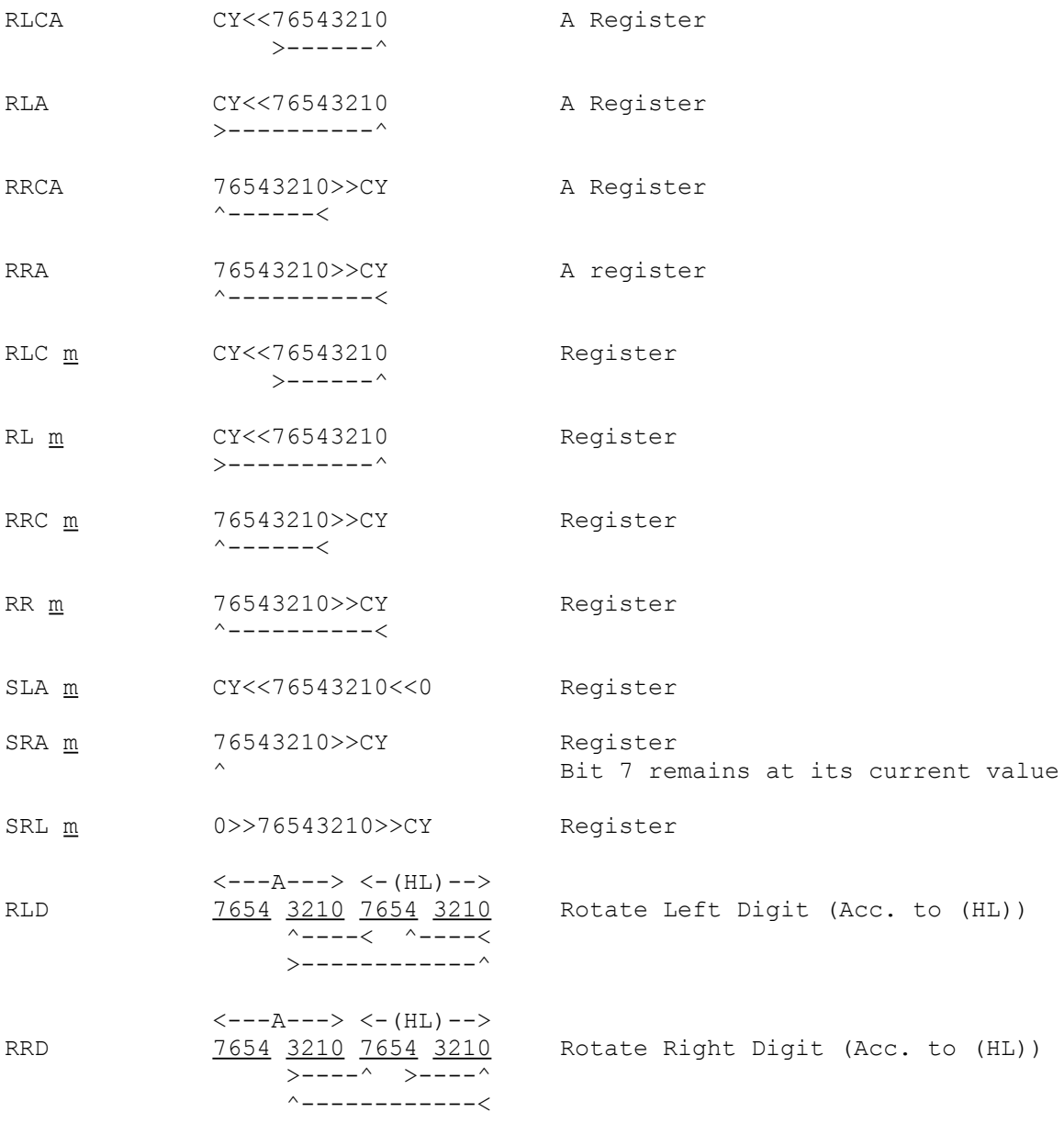
	B=B-1		10100011	A3				B->A8~15
	HL=HL+1							
OTIR	(C)=(HL)	X|XXXX|X	11101101	ED	2	5	21	C->A0~7
	B=B-1		10110011	B3		B<>0		B->A8~15
	HL=HL+1				2	4	16	
	Repeat until B=0					B=0		
OUTD*	(C)=(HL)	X|XXXX|X	11101101	ED	2	4	16	C->A0~7
	B=B-1		10110011	B3				B->A8~15
	HL=HL-1							
OTDR	(C)=(HL)	X|XXXX|X	11101101	ED	2	5	21	C->A0~7
	B=B-1		10110011	BB		B<>0		B->A8~15
	HL=HL-1				2	4	16	
	Repeat until B=0					B=0		

***** If the result of B-1 is zero the Z flag is set, otherwise it is reset

See Symbolic Notation for a description of symbols used in the table.

Shift and Rotate Operations

The following diagrams show how the various shift functions affect their operands. All shifts operate on an 8 bit operand, except RRD and RLD which shift data in 4 bit chunks between the Accumulator and the contents of the address pointed to by HL.



Interrupt Modes

Non Maskable Interrupt

The Non-Maskable interrupt cannot be masked in software. On recognition of NMI low the CPU executes a Call to location 0066H. On completion of the interrupt handler the CPU should execute a RETN instruction to continue operation.

Maskable Interrupt

The maskable interrupt may be enabled or disabled by use of the DI and EI instructions. On receipt of an interrupt the CPU takes action dependant on the current interrupt mode. The interrupt mode is set up by the IM instructions to mode 0, 1, or 2.

Interrupt Mode 0

In this mode the interrupting device places an instruction on the data bus. This is normally a Restart Instruction which will intitiate a call to on of the Restart vectors in the first page of memory.

Interrupt Mode 1

In this mode, on receiving an interrupt, the processor jumps to location 0038H. The CPU should execute a RETI (or EI - RET) instruction on completion of the interrupt routine.

Interrupt Mode 2

In this mode the interrupting device places an 8 bit vector on the bus. The CPU uses this as the bottom byte of a two byte address where the upper byte is formed from the I register. The contents of this address contain a 16 bit address which is the location of the interrupt routine.

Interrupt Enable/Disable Operation

There are two interrupt enable flags, IFF1 and IFF2. IFF1 represents the state of the Maskable Interrupt. IFF2 holds IFF1 during a non-maskable interrupt service routine.

Action	**IFF1**	**IFF2**	**Comments**
CPU Reset	0	0	Maskable interrupt disabled
DI execution	0	0	Maskable interrupt disabled
EI execution	1	1	Maskable interrupt enabled
LD A,I execution	.	.	IFF2 >> Parity flag
LD A,R execution	.	.	IFF2 >> Parity flag
Accept NMI	0	IFF1	Maskable Interrupt Disabled
RETN executed	IFF2	.	Completion of NMI service routine

Register -	000 -	B
	001 -	C
	010 -	D
	011 -	E
	100 -	H
	101 -	L
	111 -	A

IFF	Content of the Interrupt Flip Flop

Register Pair	00 - BC
	01 - DE
	10 - HL*
	11 - SP

* - If the instruction refers to an operation on IX or IY then the code 10 refers to that register, not to HL.

Register Pair	00 - BC 01 - DE 10 - HL 11 - AF

S is any of r,n,(HL),(IX+D),(IY+D) as shown for ADD instruction. The indicated bits replace the **000** in the ADD instruction.

Flags are :	7	S	Sign flag, set if the MSB of the result is 1.
	6	Z	Zero flag, set if the result is 0.
	4	H	Half carry flag, H=1 if the result of the operation caused a carry into, or a borrow from, bit 4 of the accumulator.
	2	P	Parity or Overflow flag, Logical operations set this flag with the parity of the Result, arithmetic operations set it with the overlflow of the result.
	1	N	Add/Subtract flag. N=1 if the previous operation was a subtract.
	0	C	Carry/Link flag, C=1 if the operation produced a carry from the MSB of the operand or result.

M is any of r,(HL),(IX+D),(IY+D) as shown for INC instruction. DEC same format and replace **100** with **101** in opcode.

M is any of r,(HL),(IX+D),(IY+D) as shown for RLC Instruction. Use same format and replace **000** with codes shown in opcode.

b - Bit Tested	000 - 0
	001 - 1
	010 - 2
	011 - 3
	100 - 4
	101 - 5
	110 - 6
	111 - 7

CC - Condition	000 - NZ	Non-Zero
	001 - Z	Zero
	010 - NC	No Carry
	011 - C	Carry
	100 - PO	Parity Odd
	101 - PE	Parity Even
	110 - P	Sign Positive
	111 - M	Sign Negative

P - Restart vector - 000 - Calls routine located at address 00H
001 - Calls routine located at address 08H
010 - Calls routine located at address 10H
011 - Calls routine located at address 18H
100 - Calls routine located at address 20H
101 - Calls routine located at address 28H
110 - Calls routine located at address 30H
111 - Calls routine located at address 38H